OCR co-design, SoC methods
SoC for HPC: OS / Runtimes

Josh Fryman

This work funded in part by US Government Contracts No.# 7078611, 7216501 and B608115.
SoC Methodology and Extreme Scale Challenges

1. System power & energy (on-die)
2. Efficient memory subsystem
3. Programmability: O(B) cores
4. Execution model
5. Resiliency
6. $$ Cost and affordability
7. System efficiency 20+ %
8. IO and Storage support

Soc Methodology brings:
• Menu of IP blocks
  • Pick your core
  • Pick your I/O
  • Pick memory pieces
• Focus on features, trade-offs
  • Quantity
  • Connectivity
Energy: double-precision op vs. moving 64 bits

BW tapering and data locality should remain the foremost consideration

SoC impact

Compute double OP

On Die Memory

SoC impact

Data Movement

SoC impact
Open Community Runtime (OCR)

Multi-party collaboration: Intel, Rice, UIUC, UCSD, PNNL

- Provide effective abstraction for diverse hardware (hetero-ISA ready)
- Typify future task-based execution models
- Handle large-scale parallelism efficiently and dynamically
- Provide user-perspective application-transparent resiliency
- Maintain a separation of concerns (application/scheduling/resources)
- Open source (encourage collaboration) http://xstack.exascale-tech.com

- OCR is X-Stack Traleika Glacier project’s implementation for this revolutionary run-time prototype
- FFWD-2 is extending OCR with legacy support, re-factored applications, and re-factoring guides, templates, and tools
Runtime Design Principles of OCR

Modular
Extensible
Adaptable
Tunable

SoC agnostic
SoC aware
Co-Design Opportunities via Software Analysis

Standard compute kernel source (instruction stream)

Generate Result Set 1

Send #1

Generate Result Set 2

parallel_for (..) {}
serial_ops (..) {}
[etc]

Micro-thread-like operation sequence

 Need for stalls and/or ordering at these points depends on programming model and algorithm . . .
OCR and SoC Methodology: Co-Design
SoC Methodology: What makes sense?

Pros:
• Tetris with IP blocks
  • Customize to your workload
  • Small “design team” to combine*
• Focus on what knobs matter
  • Combinations have impact
  • “Good enough” components
• Cost analysis is clear
  • Highly tuned and specialized for {app}
  • ROI based on your workloads

Cons:
• Tetris with IP blocks
  • Primitive Pete and just a hammer
  • Tweaks == not-small “design team”
• You only have a few knobs
  • They had better matter
  • May not be “good enough”
• Cost analysis is tricky
  • Volume determines $$$
  • Validation dominates TTM